MODULE NAME:	MODULE CODE:
PROGRAMMING 1B	PROG6112

ASSESSMENT TYPE:	EXAMINATION (PAPER ONLY)
TOTAL MARK ALLOCATION:	120 MARKS
TOTAL HOURS:	2 HOURS (+10 minutes reading time)

SETUP TIME – SPECIAL INSTRUCTIONS:

1. *For practical IT tests or exams written on campus, the usual reading time is replaced by an **additional 30-minute setup time** allocated for setup, saving and upload activities.*
2. *Students are allowed to make notes during the 30-minute setup time.*
3. *Students are allowed to **start working** on their practical solutions as soon as the 30-minute setup time starts.*

INSTRUCTIONS:

1. *Please adhere to all instructions in the assessment booklet.*
2. *Independent work is required.*
3. *Ten minutes is dedicated to reading time before the start of the assessment. You may make notes on your question paper, but not in your answer sheet. Calculators may not be used during reading time.*
4. *You may not leave the assessment venue during reading time, or during the first hour or during the last 15 minutes of the assessment.*
5. *Ensure that your name is on all pieces of paper or books that you will be submitting. Submit all the pages of this assessment's question paper as well as your answer script.*
6. *Answer all the questions on the answer sheets or in answer booklets provided. The phrase 'END OF PAPER' will appear after the final set question of this assessment.*
7. *Remember to work at a steady pace so that you are able to complete the assessment within the allocated time. Use the mark allocation as a guideline as to how much time to spend on each section.*

Additional instructions:

1. *This is an OPEN BOOK assessment.*
2. *Students can use lab computers or their personal devices to connect to Azure Lab Services and to complete the assessment. If students choose to use their own devices they must ensure that they are able to connect to campus networks, Azure Lab Services and the LMS in advance of the assessment sitting. No campus assistance is available during the assessment to troubleshoot problems with personal devices.*
3. *For open book assessments the students may have open access to all resources inclusive of notes, books (hardcopy and e-books) and the internet. These resources may be accessed as hard copies or as electronic files on electronic devices. All electronic devices batteries must be fully charged before the assessment as no charging of devices will be permitted during the sitting of the assessment. The IIE and its sites of delivery accept no liability for the loss or damage incurred to electronic devices used during open book assessments.*
4. *Answer All Questions.*
5. *Instructions for submitting your assessment:*

- *Use of good programming practice and comments in code is compulsory.*
- *Save your solution/project in the designated space for this module.*
- *Save all files (including any source code files, template files, design files, image files, text files, database files, etc.) within the designated space.*
- *Do NOT save zipped (archive) files in the designated space unless specifically instructed to do so.*
- ***Important:** Upon completion of your assessment, you must save and close all your open files before submitting your work. You will submit your assessment on the LMS page for this module.*
- ***To complete your submission:** Create a document in MS Word or Notepad. The document name must follow the format shown here:*
- ***StudentNumber_ModuleCode_Test.** E.g., if your student number is 12345 and you are writing a Test for the module PROG6112, create a document named **12345_PROG6112_Exam.***
- *In this document include the following: Your student number, the module code, and **the link to designated space** where you saved your practical work. If you are required to include any written answers, also include type these answers in the same document.*
- *Submit this document in the LMS, using the Exam submission link for this module.*

GENERAL REQUIREMENTS **Marks: 10)**

Writing maintainable code in industry is vital, therefore, you are generally required to:

- Follow good programming practices:
  - Variable types correct;
  - Variable scope correct; and
  - Class and method naming standards correct.
- Insert comments to show logic and design for the support of code maintainability.
- Code efficiently. Redundant code must be avoided.
- Spend 15 minutes ensuring that your programs meet the general criteria below.

Requirement	**Maximum Mark**	**Examiner's Mark**	**Moderator's Mark**
Good Programming Practice • Not followed at all = 0 • Average – Minor changes required = 1 – 2 • Excellent – No changes required = 3	(3)		
Code Efficiency • Poor – Much code is duplicated = 0 • Average – Some redundant code = 1 • Excellent – Code very maintainable = 2	(2)		
Comment Statements • Poor – No comments = 0 • Average – Some comments = 1 – 2 • Excellent – All necessary comments given to show logic = 3	(3)		
Program(s) compile and execute • No – Many changes required = 0 • Average – Minor changes required = 1 • Yes – No changes required = 2	(2)		
TOTAL MARKS	**10**		

Question 1 **(Marks: 50)**

Q.1.1 Write a Java application to display the number of movie tickets sold for two different movies at a local cinema. The movie report generated will display the movie name and the total sales for the months of January, February, and March of 2024. The table below shows the movie and ticket sales for each month:

	JAN	FEB	MAR
Napoleon	3000	1500	1700
Oppenheimer	3500	1200	1600

Sample screenshot

```
MOVIE TICKET SALES REPORT - 2024

                        JAN             FEB             MAR
-------------------------------------------------------------------------
Napoleon                3000            1500            1700
Oppenheimer             3500            1200            1600

Total movie ticket sales for Napoleon 6200
Total movie ticket sales for Oppenheimer 6300

Top performing movie: Oppenheimer
```

Using single and two dimensional arrays produce the movie report and include the total movie ticket sales for each movie.

In your solution also include the top performing movie.

Q.1.2 Make use of a class named MovieTickets that contains methods to calculate the total movie sales and the top performing movie. The MovieTickets class must implement an IMovieTickets interface that contains the following:

```
public interface IMovieTickets {
    int TotalMovieSales(int[] movieTicketSales);
    String TopMovie(String[] movies, int[] totalSales);
}
```

Q.1.3	You are required to write unit tests for the application. Create a test package within the application you created, which will contain the necessary unit tests.

You are required to write the following unit tests:

Test Name	Test Purpose
CalculateTotalSales_ReturnsExpectedTotalSales	To supply the movie ticket sales to the total movie sales method. The test will determine that the correct total sales value is returned from the Total Movie Sales method.
TopMovieSales_ReturnsTopMovie	The test will determine the top performing movie.

Question 1 Mark Allocation	Levels of Achievement				Feedback
	Excellent	Good	Developing	Poor	
	Score Ranges Per Level (½ marks possible)				
Declaration and Population of single and two-dimensional arrays.	**10**	**5-9**	**1-4**	**0**	
	Single and two dimensional arrays created correctly.	Minor changes required	Major changes required	Not provided	
Printing of rows and columns in the report.	**5**	**3-4**	**1-2**	**0**	
	The report has been created successfully using rows and columns.	Minor changes required	Major changes required	Not provided	
Printing and calculating the total movie sales for each movie.	**10**	**5-9**	**1-4**	**0**	

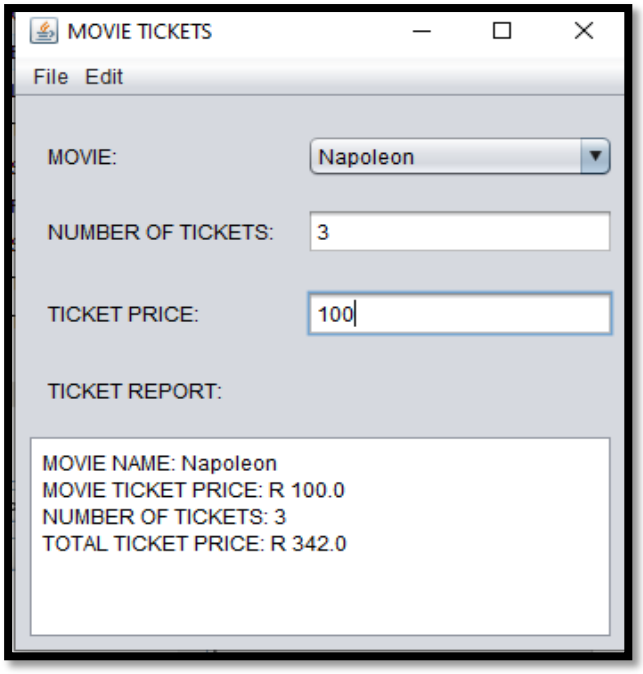
	Calculate the total sales for each movie	Minor changes required	Major changes required	Not provided	
Determine and display the top performing movie.	**5**	**3-4**	**1-2**	**0**	
	Correct calculation and printing of top performing movie	Minor changes required	Major changes required	Not provided	
IMovieTickets interface class created	**5**	**3-4**	**1-2**	**0**	
	IMovieTickets interface class created.	Minor changes required	Major changes required	Not provided	
MovieTickets class created that implements the IMovieTickets class with TotalMovieSales and TopMovie methods	**5**	**3-4**	**1-2**	**0**	
	MovieTickets created that implements the IMovieTickets class with	Minor changes required	Major changes required	Not provided	

	required methods				
Calculate Total Sales Returns Expected Total Sales unit test created that tests the required functionality	**5**	**3-4**	**1-2**	**0**	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	
Top Movie Sales Returns Top Movie unit test created that tests the required functionality	**5**	**3-4**	**1-2**	**0**	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	

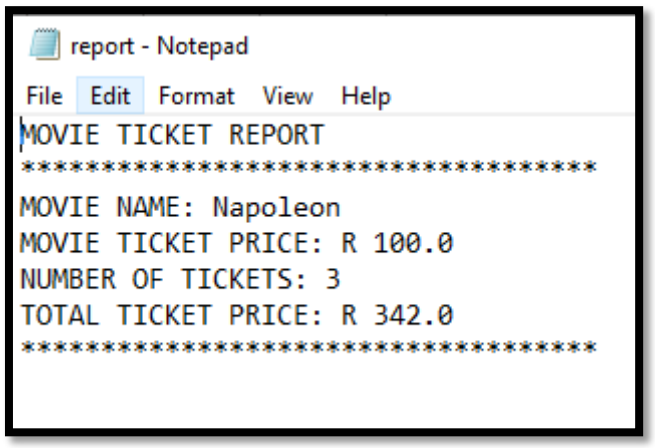
Question 2 **(Marks: 60)**

Create a Java GUI application that will process the movie tickets purchased for a customer at a local cinema. The application must capture all the required input and display the movie ticket sales report which includes the VAT total amount.

Sample Screenshot



Q.2.1	On the form create one (1) combo box for the movie selection. The movies available are Napoleon, Oppenheimer, and Damsel. You are also required to create two (2) text fields to capture the number of movie tickets and the movie ticket price. Also create a read only text area to display the movie sales report.
Q.2.2	A menu system is required for the user to interact with. The following menu items are required: **File** • Exit **Tools** • Process • Clear

Q.2.3	When the Exit menu item is clicked, close the application.
Q.2.4	When the Process menu item is clicked, capture the selected movie, number of tickets, and the movie ticket price, and display the movie ticket sales report. The total movie ticket price must include the VAT amount of 14%.
Q.2.5	After the movie ticket sales report has been generated save the movie sales to a text file named report.txt. **Sample Screenshot** 
Q.2.6	When the Clear Menu item is clicked set the text fields and text area to the default state.
Q.2.7	Make use of a class named MovieTickets that contains methods to calculate the total ticket price and validate the data. Only valid data should be captured in the application; the validation rules are:

VALIDATION	RULE
Movie Ticket Name	Cannot be empty
Movie Ticket Price	Cannot be less than or equal to zero (0)
Number of Movie Tickets	Cannot be less than or equal to zero (0)

Q.2.8 The MovieTickets class must implement an IMovieTickets interface that contains the following:

```
public interface IMovieTickets {
   double CalculateTotalTicketPrice(int numberOfTickets, double ticketPrice);
   boolean ValidateData(MovieTicketData movieTicketData);
}
```

Q.2.9 You are required to write unit tests for the application. Create a test package within the application you created, which will contain the necessary unit tests.

You are required to write the following unit tests:

Test Name	Test Purpose
CalculateTotalTicketPrice_CalculatedSuccessfully	To supply the number of tickets and ticket price to the Calculate Total Ticket Price method. The test will determine that the correct total price value is returned.
Validation Tests	You are only required to write one unit test to prove that the validation is performing as expected. Examples can include movie name is not valid, number of tickets is not valid or validating ticket price is not valid.

Question 2 Mark Allocation	Levels of Achievement				Feedback
	Excellent	Good	Developing	Poor	
	Score Ranges Per Level (½ marks possible)				
GUI form created with all objects	**9-10**	**5-8**	**1-4**	**0**	
	GUI form created with all objects	Minor changes are required.	Major changes are required.	Not provided.	
Populating the combo box with the correct values	**5**	**2-4**	**1**	**0**	
	Populating the combo box with the correct values	Minor changes required.	Major changes required.	Not provided.	
Creating the menu system with required menu items	**5**	**3-4**	**1-2**	**0**	
	Creating the menu system with required menu items	Minor changes are required.	Major changes are required.	Not provided.	
Report produced as per sample	**5**	**2-4**	**1**	**0**	

	Report produced as per sample	Minor changes required	Major changes required.	Not provided.	
File saved correctly with the report contents.	**5**	**2-4**	**1**	**0**	
	File saved correctly	Minor changes required.	Major changes required.	No Output	
IMovieTickets interface created	**5**	**2-4**	**1**	**0**	
	IMovieTickets interface created	Minor changes required.	Major changes required.	No Output	
Movie Tickets class created that implements IMovieTickets with required methods	**5**	**2-4**	**1**	**0**	
	Movie Tickets class created with required methods	Minor changes required.	Major changes required.	No Output	

Total Movie Ticket Price calculated correctly	**5**	**2-4**	**1**	**0**	
	Total Movie Price with VAT amount calculated correctly	Minor changes required.	Major changes required.	No Output	
Validation performed correctly	**5**	**2-4**	**1**	**0**	
	Validation performed correctly	Minor changes required.	Major changes required.	No Output	
Calculate Total Ticket Price Calculated Successfully unit test created that tests the required functionality	**5**	**2-4**	**1-2**	**0**	
	Unit test created that tests the required functionality	Minor changes required	Major changes required	Not provided	
Any validation test to prove that the data captured is valid	**5**	**2-4**	**1**	**0**	

	Validation unit test created that tests the required functionality	Minor changes required.	Major changes required.	No Output	

END OF PAPER